W3C Recommendation

RDF/XML Syntax Specification (Revised)

W3C Recommendation 10 February 2004

This version:
http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/
Latest version:
http://www.w3.org/TR/rdf-syntax-grammar/
Previous version:
http://www.w3.org/TR/2003/PR-rdf-syntax-grammar-20031215/
Editor:
Dave Beckett (University of Bristol)
Series editor:
Brian McBride (Hewlett Packard Labs)

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

See also translations.

Abstract

The Resource Description Framework (RDF) is a general-purpose language for representing information in the Web.

This document defines an XML syntax for RDF called RDF/XML in terms of Namespaces in XML, the XML Information Set

and XML Base. The formal grammar for the syntax is annotated with actions generating triples of the RDF graph as defined in RDF Concepts and Abstract Syntax. The triples are written using the N-Triples RDF graph serializing format which enables more precise recording of the mapping in a machine processable form. The mappings are recorded as tests cases, gathered and published in RDF Test Cases.

Status of this Document

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

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

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

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

A list of implementations is available.

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

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

Table of Contents

Appendices

---

1 Introduction

This document defines the XML [XML] syntax for RDF graphs which was originally defined in the RDF Model & Syntax [RDF-MS] W3C Recommendation. Subsequent implementations of this syntax and comparison of the resulting RDF graphs have shown that there was ambiguity — implementations generated different graphs and certain syntax forms were not widely implemented.

This document revises the original RDF/XML grammar in terms of XML Information Set [INFOSET] information items which moves away from the rather low-level details of XML, such as particular forms of empty elements. This allows the grammar to be more precisely recorded and the mapping from the XML syntax to the RDF Graph more clearly shown. The mapping to the RDF graph is done by emitting statements in the form defined in the N-Triples section of RDF Test Cases [RDF-TESTS] which creates an RDF graph, that has semantics defined by RDF Semantics [RDF-SEMANTICS].

The complete specification of RDF consists of a number of documents:

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

For a longer introduction to the RDF/XML syntax with a historical perspective, see RDF: Understanding the Striped RDF/XML Syntax [STRIPEDRDF].

2 An XML Syntax for RDF

This section introduces the RDF/XML syntax, describes how it encodes RDF graphs and explains this with examples. If there is any conflict between this informal description and the formal description of the syntax and grammar in sections 6 Syntax Data Model and 7 RDF/XML Grammar, the latter two sections take precedence.

2.1 Introduction

The RDF Concepts and Abstract Syntax [RDF-CONCEPTS] defines the RDF Graph data model (Section 3.1) and the RDF Graph abstract syntax (Section 6). Along with the RDF Semantics [RDF-SEMANTICS] this provides an abstract syntax with a formal semantics for it. The RDF graph has nodes and labeled directed *arcs* that link pairs of nodes and this is represented as a set of RDF triples where each triple contains a *subject node*, *predicate* and *object node*. Nodes are RDF URI references, RDF literals or are blank nodes. Blank nodes may be given a document-local, non-RDF URI references identifier called a blank node identifier. Predicates are RDF URI references and can be interpreted as either a relationship between the two nodes or as defining an attribute value (object node) for some subject node.

In order to encode the graph in XML, the nodes and predicates have to be represented in XML terms — element names, attribute names, element contents and attribute values. RDF/XML uses XML QNames as defined in Namespaces in XML [XML-NS] to represent RDF URI references. All QNames have a namespace name which is a URI reference and a short local name. In addition, QNames can either have a short prefix or be declared with the default namespace declaration and have none (but still have a namespace name)

The RDF URI reference represented by a QName is determined by appending the local name part of the QName after the namespace name (URI reference) part of the QName. This is used to shorten the RDF URI references of all predicates and some nodes. RDF URI references identifying subject and object nodes can also be stored as XML attribute values. RDF literals, which can only be object nodes, become either XML element text content or XML attribute values.

A graph can be considered a collection of paths of the form node, predicate arc, node, predicate arc, node, predicate arc, ... node which cover the entire graph. In RDF/XML these turn into sequences of elements inside elements which alternate between elements for nodes and predicate arcs. This has been called a series of node/arc stripes. The node at the start of the sequence turns into the outermost element, the next predicate arc turns into a child element, and so on. The stripes generally start at the top of an RDF/XML document and always begin with nodes.

Several RDF/XML examples are given in the following sections building up to complete RDF/XML documents. Example 7 is

the first complete RDF/XML document.

2.2 Node Elements and Property Elements

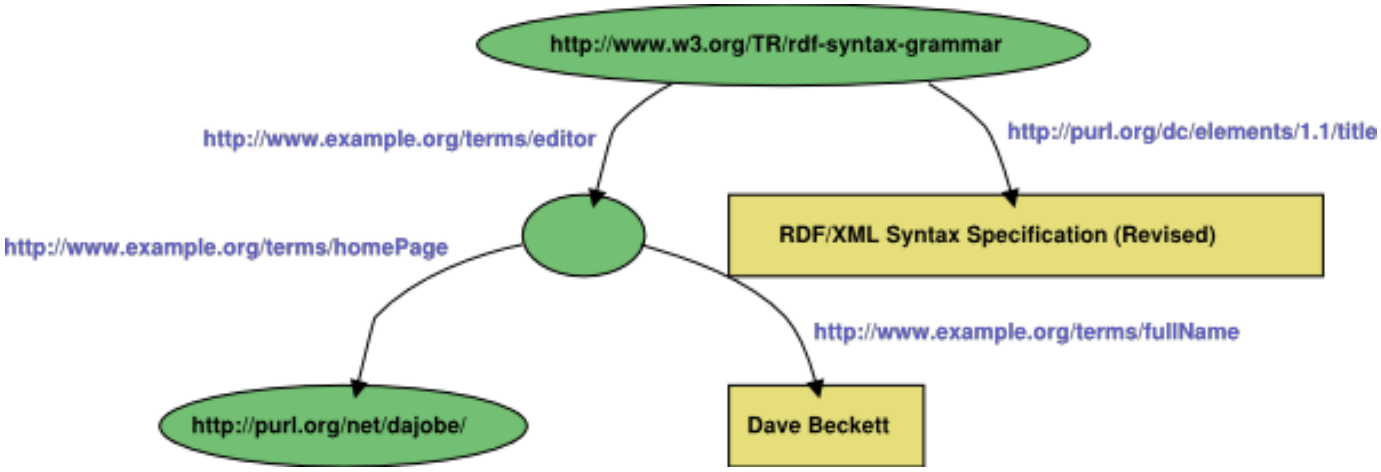

Figure 1: Graph for RDF/XML Example (SVG version)

An RDF graph is given in Figure 1 where the nodes are represented as ovals and contain their RDF URI references where they have them, all the predicate arcs are labeled with RDF URI references and plain literal nodes have been written in rectangles.

If we follow one node, predicate arc ... , node path through the graph shown in Figure 2:

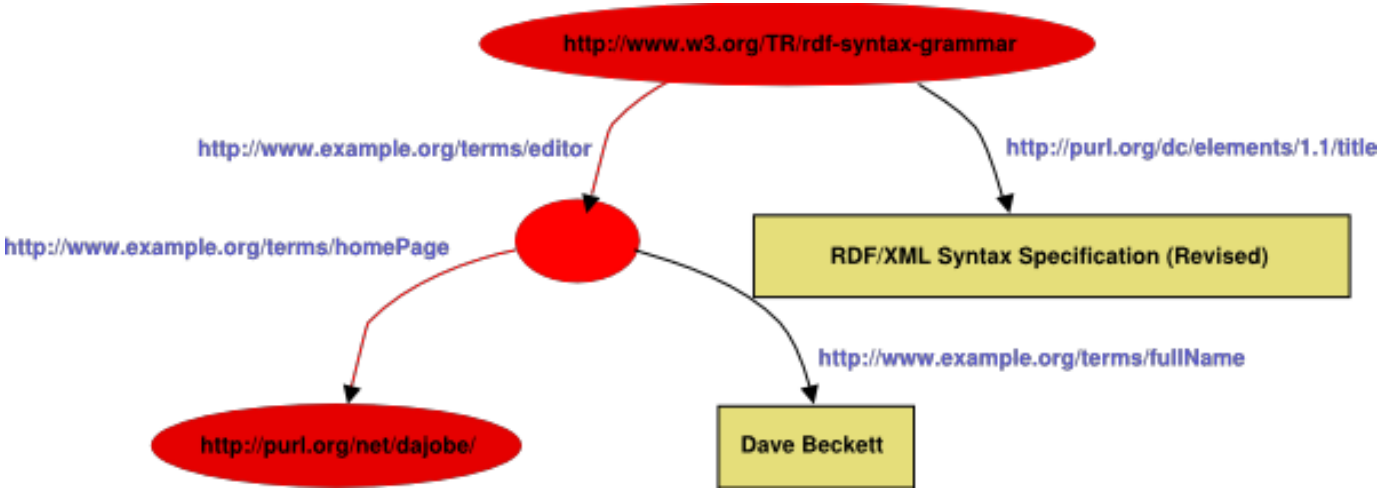

Figure 2: One Path Through the Graph (SVG version)

The left hand side of the Figure 2 graph corresponds to the node/predicate arc stripes:

1. Node with RDF URI reference `http://www.w3.org/TR/rdf-syntax-grammar`
2. Predicate Arc labeled with RDF URI reference `http://example.org/terms/editor`
3. Node with no RDF URI reference
4. Predicate Arc labeled with RDF URI reference `http://example.org/terms/homePage`
5. Node with RDF URI reference `http://purl.org/net/dajobe/`

In RDF/XML, the sequence of 5 nodes and predicate arcs on the left hand side of Figure 2 corresponds to the usage of five XML elements of two types, for the graph nodes and predicate arcs. These are conventionally called *node elements* and *property elements* respectively. In the striping shown in Example 1, `rdf:Description` is the node element (used three times for the three nodes) and `ex:editor` and `ex:homePage` are the two property elements.

Example 1: Striped RDF/XML (nodes and predicate arcs)

```
<rdf:Description>
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description>
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

The Figure 2 graph consists of some nodes that are RDF URI references (and others that are not) and this can be added to the RDF/XML using the `rdf:about` attribute on node elements to give the result in Example 2:

Example 2: Node Elements with RDF URI references added

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
```

```
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

Adding the other two paths through the Figure 1 graph to the RDF/XML in Example 2 gives the result in Example 3 (this example fails to show that the blank node is shared between the two paths, see 2.10):

Example 3: Complete description of all graph paths

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
</rdf:Description>
```

2.3 Multiple Property Elements

There are several abbreviations that can be used to make common uses easier to write down. In particular, it is common that a subject node in the RDF graph has multiple outgoing predicate arcs. RDF/XML provides an abbreviation for the

corresponding syntax when a node element about a resource has multiple property elements. This can be abbreviated by using multiple child property elements inside the node element describing the subject node.

Taking Example 3, there are two node elements that can take multiple property elements. The subject node with URI reference `http://www.w3.org/TR/rdf-syntax-grammar` has property elements `ex:editor` and `ex:title` and the node element for the blank node can take `ex:homePage` and `ex:fullName`. This abbreviation gives the result shown in Example 4 (this example does show that there is a single blank node):

Example 4: Using multiple property elements on a node element

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
</rdf:Description>
```

2.4 Empty Property Elements

When a predicate arc in an RDF graph points to an object node which has no further predicate arcs, which appears in RDF/XML as an empty node element `<rdf:Description rdf:about="..."> </rdf:Description>` (or `<rdf:Description rdf:about="..." />`) this form can be shortened. This is done by using the RDF URI reference of the object node as the value of an XML attribute `rdf:resource` on the containing property element and making the property element empty.

In this example, the property element `ex:homePage` contains an empty node element with the RDF URI reference `http://purl.org/net/dajobe/`. This can be replaced with the empty property element form giving the result shown in Example 5:

Example 5: Empty property elements

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
```

```
  <ex:editor>
    <rdf:Description>
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
</rdf:Description>
```

2.5 Property Attributes

When a property element's content is string literal, it may be possible to use it as an XML attribute on the containing node element. This can be done for multiple properties on the same node element only if the property element name is not repeated (required by XML — attribute names are unique on an XML element) and any in-scope `xml:lang` on the property element's string literal (if any) are the same (see Section 2.7) This abbreviation is known as a *Property Attribute* and can be applied to any node element.

This abbreviation can also be used when the property element is `rdf:type` and it has an `rdf:resource` attribute the value of which is interpreted as a RDF URI reference object node.

In Example 5:, there are two property elements with string literal content, the `dc:title` and `ex:fullName` property elements. These can be replaced with property attributes giving the result shown in Example 6:

Example 6: Replacing property elements with string literal content into property attributes

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                 dc:title="RDF/XML Syntax Specification (Revised)">
  <ex:editor>
    <rdf:Description ex:fullName="Dave Beckett">
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

2.6 Completing the Document: Document Element and XML Declaration

To create a complete RDF/XML document, the serialization of the graph into XML is usually contained inside an `rdf:RDF` XML element which becomes the top-level XML document element. Conventionally the `rdf:RDF` element is also used to declare the XML namespaces that are used, although that is not required. When there is only one top-level node element inside `rdf:RDF`, the `rdf:RDF` can be omitted although any XML namespaces must still be declared.

The XML specification also permits an XML declaration at the top of the document with the XML version and possibly the XML content encoding. This is optional but recommended.

Completing the RDF/XML could be done for any of the correct complete graph examples from Example 4 onwards but taking the smallest Example 6 and adding the final components, gives a complete RDF/XML representation of the original Figure 1 graph in Example 7:

Example 7: Complete RDF/XML description of Figure 1 graph (example07.rdf output example07.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                   dc:title="RDF/XML Syntax Specification (Revised)">
    <ex:editor>
      <rdf:Description ex:fullName="Dave Beckett">
        <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
      </rdf:Description>
    </ex:editor>
  </rdf:Description>
</rdf:RDF>
```

It is possible to omit `rdf:RDF` in Example 7 above since there is only one `rdf:Description` inside `rdf:RDF` but this is not shown here.

2.7 Languages: `xml:lang`

RDF/XML permits the use of the `xml:lang` attribute as defined by 2.12 Language Identification of XML 1.0 [XML] to allow the identification of content language. The `xml:lang` attribute can be used on any node element or property element to indicate that the included content is in the given language. Typed literals which includes XML literals are not affected by this attribute.

The most specific in-scope language present (if any) is applied to property element string literal content or property attribute values. The `xml:lang=""` form indicates the absence of a language identifier.

Some examples of marking content languages for RDF properties are shown in Example 8:

Example 8: Complete example of `xml:lang` (example08.rdf output example08.nt)

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
    <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
    <dc:title xml:lang="en">RDF/XML Syntax Specification (Revised)</dc:title>
    <dc:title xml:lang="en-US">RDF/XML Syntax Specification (Revised)</dc:title>
  </rdf:Description>

  <rdf:Description rdf:about="http://example.org/buecher/baum" xml:lang="de">
    <dc:title>Der Baum</dc:title>
    <dc:description>Das Buch ist außergewöhnlich</dc:description>
    <dc:title xml:lang="en">The Tree</dc:title>
  </rdf:Description>
</rdf:RDF>
```

2.8 XML Literals: `rdf:parseType="Literal"`

RDF allows XML literals ([RDF-CONCEPTS] Section 5, *XML Content within an RDF graph*) to be given as the object node of a predicate. These are written in RDF/XML as content of a property element (not a property attribute) and indicated using the `rdf:parseType="Literal"` attribute on the containing property element.

An example of writing an XML literal is given in Example 9 where there is a single RDF triple with the subject node RDF URI reference `http://example.org/item01`, the predicate RDF URI reference `http://example.org/stuff/1.0/prop` (from `ex:prop`) and the object node with XML literal content beginning `a:Box`.

Example 9: Complete example of `rdf:parseType="Literal"` (example09.rdf output example09.nt)

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

```
  <rdf:Description rdf:about="http://example.org/item01">
    <ex:prop rdf:parseType="Literal"
             xmlns:a="http://example.org/a#"><a:Box required="true">
          <a:widget size="10" />
          <a:grommit id="23" /></a:Box>
    </ex:prop>
  </rdf:Description>
</rdf:RDF>
```

2.9 Typed Literals: `rdf:datatype`

RDF allows typed literals to be given as the object node of a predicate. Typed literals consist of a literal string and a datatype RDF URI reference. These are written in RDF/XML using the same syntax for literal string nodes in the property element form (not property attribute) but with an additional `rdf:datatype="`*datatypeURI*`"` attribute on the property element. Any RDF URI reference can be used in the attribute.

An example of an RDF typed literal is given in Example 10 where there is a single RDF triple with the subject node RDF URI reference `http://example.org/item01`, the predicate RDF URI reference `http://example.org/stuff/1.0/size` (from `ex:size`) and the object node with the typed literal ("123", `http://www.w3.org/2001/XMLSchema#int`) to be interpreted as an W3C XML Schema [XML-SCHEMA2] datatype int.

Example 10: Complete example of `rdf:datatype` (example10.rdf output example10.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://example.org/item01">
    <ex:size rdf:datatype="http://www.w3.org/2001/XMLSchema#int">123</ex:size>
  </rdf:Description>
</rdf:RDF>
```

2.10 Identifying Blank Nodes: `rdf:nodeID`

Blank nodes in the RDF graph are distinct but have no RDF URI reference identifier. It is sometimes required that the same graph blank node is referred to in the RDF/XML in multiple places, such as at the subject and object of several RDF triples. In

this case, a *blank node identifier* can be given to the blank node for identifying it in the document. Blank node identifiers in RDF/XML are scoped to the containing XML Information Set document information item. A blank node identifier is used on a node element to replace `rdf:about="`*RDF URI reference*`"` or on a property element to replace `rdf:resource="`*RDF URI reference*`"` with `rdf:nodeID="`*blank node identifier*`"` in both cases.

Taking Example 7 and explicitly giving a blank node identifier of `abc` to the blank node in it gives the result shown in Example 11. The second `rdf:Description` property element is about the blank node.

Example 11: Complete RDF/XML description of graph using `rdf:nodeID` identifying the blank node (example11.rdf output example11.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                   dc:title="RDF/XML Syntax Specification (Revised)">
    <ex:editor rdf:nodeID="abc"/>
  </rdf:Description>

  <rdf:Description rdf:nodeID="abc"
                   ex:fullName="Dave Beckett">
    <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
  </rdf:Description>
</rdf:RDF>
```

2.11 Omitting Blank Nodes: `rdf:parseType="Resource"`

Blank nodes (not RDF URI reference nodes) in RDF graphs can be written in a form that allows the `<rdf:Description>` `</rdf:Description>` pair to be omitted. The omission is done by putting an `rdf:parseType="Resource"` attribute on the containing property element that turns the property element into a property-and-node element, which can itself have both property elements and property attributes. Property attributes and the `rdf:nodeID` attribute are not permitted on property-and-node elements.

Taking the earlier Example 7, the contents of the `ex:editor` property element could be alternatively done in this fashion to give the form shown in Example 12:

Example 12: Complete example using `rdf:parseType="Resource"` (example12.rdf output example12.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                   dc:title="RDF/XML Syntax Specification (Revised)">
    <ex:editor rdf:parseType="Resource">
      <ex:fullName>Dave Beckett</ex:fullName>
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
    </ex:editor>
  </rdf:Description>
</rdf:RDF>
```

2.12 Omitting Nodes: Property Attributes on an empty Property Element

If all of the property elements on a blank node element have string literal values with the same in-scope `xml:lang` value (if present) and each of these property elements appears at most once and there is at most one `rdf:type` property element with a RDF URI reference object node, these can be abbreviated by moving them to be property attributes on the containing property element which is made an empty element.

Taking the earlier Example 5, the `ex:editor` property element contains a blank node element with two property elements `ex:fullname` and `ex:homePage`. `ex:homePage` is not suitable here since it does not have a string literal value, so it is being *ignored* for the purposes of this example. The abbreviated form removes the `ex:fullName` property element and adds a new property attribute `ex:fullName` with the string literal value of the deleted property element to the `ex:editor` property element. The blank node element becomes implicit in the now empty `ex:editor` property element. The result is shown in Example 13.

Example 13: Complete example of property attributes on an empty property element (example13.rdf output example13.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                   dc:title="RDF/XML Syntax Specification (Revised)">
    <ex:editor ex:fullName="Dave Beckett" />
```

```
    <!-- Note the ex:homePage property has been ignored for this example -->
  </rdf:Description>
</rdf:RDF>
```

2.13 Typed Node Elements

It is common for RDF graphs to have `rdf:type` predicates from subject nodes. These are conventionally called *typed nodes* in the graph, or *typed node elements* in the RDF/XML. RDF/XML allows this triple to be expressed more concisely. by replacing the `rdf:Description` node element name with the namespaced-element corresponding to the RDF URI reference of the value of the type relationship. There may, of course, be multiple `rdf:type` predicates but only one can be used in this way, the others must remain as property elements or property attributes.

The typed node elements are commonly used in RDF/XML with the built-in classes in the RDF vocabulary: `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:Statement`, `rdf:Property` and `rdf:List`.

For example, the RDF/XML in Example 14 could be written as shown in Example 15.

Example 14: Complete example with `rdf:type` (example14.rdf output example14.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://example.org/thing">
    <rdf:type rdf:resource="http://example.org/stuff/1.0/Document"/>
    <dc:title>A marvelous thing</dc:title>
  </rdf:Description>
</rdf:RDF>
```

Example 15: Complete example using a typed node element to replace an `rdf:type` (example15.rdf output example15.nt)

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

```
    <dc:title>A marvelous thing</dc:title>
  </ex:Document>
</rdf:RDF>
```

2.14 Abbreviating URIs: `rdf:ID` and `xml:base`

RDF/XML allows further abbreviating RDF URI references in XML attributes in two ways. The XML Infoset provides a base URI attribute `xml:base` that sets the base URI for resolving relative RDF URI references, otherwise the base URI is that of the document. The base URI applies to all RDF/XML attributes that deal with RDF URI references which are `rdf:about`, `rdf:resource`, `rdf:ID` and `rdf:datatype`.

The `rdf:ID` attribute on a node element (not property element, that has another meaning) can be used instead of `rdf:about` and gives a relative RDF URI reference equivalent to `#` concatenated with the `rdf:ID` attribute value. So for example if `rdf:ID="name"`, that would be equivalent to `rdf:about="#name"`. `rdf:ID` provides an additional check since the same *name* can only appear once in the scope of an `xml:base` value (or document, if none is given), so is useful for defining a set of distinct, related terms relative to the same RDF URI reference.

Both forms require a base URI to be known, either from an in-scope `xml:base` or from the URI of the RDF/XML document.

Example 16 shows abbreviating the node RDF URI reference of `http://example.org/here/#snack` using an `xml:base` of `http://example.org/here/` and an `rdf:ID` on the `rdf:Description` node element. The object node of the `ex:prop` predicate is an absolute RDF URI reference resolved from the `rdf:resource` XML attribute value using the in-scope base URI to give the RDF URI reference `http://example.org/here/fruit/apple`.

Example 16: Complete example using `rdf:ID` and `xml:base` for shortening URIs (example16.rdf output example16.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/stuff/1.0/"
         xml:base="http://example.org/here/">
  <rdf:Description rdf:ID="snack">
    <ex:prop rdf:resource="fruit/apple"/>
  </rdf:Description>
</rdf:RDF>
```

2.15 Container Membership Property Elements: `rdf:li` and `rdf:_n`

RDF has a set of container membership properties and corresponding property elements that are mostly used with instances of the `rdf:Seq`, `rdf:Bag` and `rdf:Alt` classes which may be written as typed node elements. The list properties are `rdf:_1`, `rdf:_2` etc. and can be written as property elements or property attributes as shown in Example 17. There is an `rdf:li` special property element that is equivalent to `rdf:_1`, `rdf:_2` in order, explained in detail in section 7.4. The mapping to the container membership properties is always done in the order that the `rdf:li` special property elements appear in XML — the document order is significant. The equivalent RDF/XML to Example 17 written in this form is shown in Example 18.

Example 17: Complex example using RDF list properties (example17.rdf output example17.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Seq rdf:about="http://example.org/favourite-fruit">
    <rdf:_1 rdf:resource="http://example.org/banana"/>
    <rdf:_2 rdf:resource="http://example.org/apple"/>
    <rdf:_3 rdf:resource="http://example.org/pear"/>
  </rdf:Seq>
</rdf:RDF>
```

Example 18: Complete example using `rdf:li` property element for list properties (example18.rdf output example18.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Seq rdf:about="http://example.org/favourite-fruit">
    <rdf:li rdf:resource="http://example.org/banana"/>
    <rdf:li rdf:resource="http://example.org/apple"/>
    <rdf:li rdf:resource="http://example.org/pear"/>
  </rdf:Seq>
</rdf:RDF>
```

2.16 Collections: `rdf:parseType="Collection"`

RDF/XML allows an `rdf:parseType="Collection"` attribute on a property element to let it contain multiple node elements. These contained node elements give the set of subject nodes of the collection. This syntax form corresponds to a set of triples connecting the collection of subject nodes, the exact triples generated are described in detail in Section 7.2.19 Production parseTypeCollectionPropertyElt. The collection construction is always done in the order that the node elements appear in the

XML document. Whether the order of the collection of nodes is significant is an application issue and not defined here.

Example 19 shows a collection of three nodes elements at the end of the `ex:hasFruit` property element using this form.

Example 19: Complete example of a RDF collection of nodes using `rdf:parseType="Collection"` (example19.rdf output example19.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://example.org/basket">
    <ex:hasFruit rdf:parseType="Collection">
      <rdf:Description rdf:about="http://example.org/banana"/>
      <rdf:Description rdf:about="http://example.org/apple"/>
      <rdf:Description rdf:about="http://example.org/pear"/>
    </ex:hasFruit>
  </rdf:Description>
</rdf:RDF>
```

2.17 Reifying Statements: `rdf:ID`

The `rdf:ID` attribute can be used on a property element to reify the triple that it generates (See section 7.3 Reification Rules for the full details). The identifier for the triple should be constructed as a RDF URI reference made from the relative URI reference `#` concatenated with the `rdf:ID` attribute value, resolved against the in-scope base URI. So for example if `rdf:ID="triple"`, that would be equivalent to the RDF URI reference formed from relative URI reference `#triple` against the base URI. Each (`rdf:ID` attribute value, base URI) pair has to be unique in an RDF/XML document, see constraint-id.

Example 20 shows a `rdf:ID` being used to reify a triple made from the `ex:prop` property element giving the reified triple the RDF URI reference `http://example.org/triples/#triple1`.

Example 20: Complete example of `rdf:ID` reifying a property element (example20.rdf output example20.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/stuff/1.0/"
         xml:base="http://example.org/triples/">
  <rdf:Description rdf:about="http://example.org/">
    <ex:prop rdf:ID="triple1">blah</ex:prop>
  </rdf:Description>
```

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

3 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

All use of string without further qualification refers to a Unicode[UNICODE] character string; a sequence of characters represented by a code point in Unicode. (Such as defined in [CHARMOD] in section 3.4 Strings).

4 RDF MIME Type, File Extension and Macintosh File Type

The Internet media type / MIME type for RDF/XML is "`application/rdf+xml`" — see RFC 3023 [RFC-3023] section 8.18.

> **Registration Note (Informative):** For the state of the MIME type registration, consult IANA MIME Media Types [IANA-MEDIA-TYPES]

It is recommended that RDF/XML files have the extension "`.rdf`" (all lowercase) on all platforms.

It is recommended that RDF/XML files stored on Macintosh HFS file systems be given a file type of "`rdf `" (all lowercase, with a space character as the fourth letter).

5 Global Issues

5.1 The RDF Namespace and Vocabulary

> **Note (Informative):** The names `aboutEach` and `aboutEachPrefix` were removed from the language and the RDF vocabulary by the RDF Core Working Group. See the resolution of issues rdfms-abouteach and rdfms-abouteachprefix for further information.

> **Note (Informative):** The names `List`, `first`, `rest` and `nil` were added for issue rdfms-seq-representation. The names `XMLLiteral` and `datatype` were added to support RDF datatyping. The name `nodeID` was added for issue rdfms-syntax-

incomplete. See the RDF Core Issues List for further information.

The RDF namespace URI reference (or namespace name) is `http://www.w3.org/1999/02/22-rdf-syntax-ns#` and is typically used in XML with the prefix `rdf` although other prefix strings may be used. The RDF Vocabulary is identified by this namespace name and consists of the following names only:

Syntax names — not concepts

```
RDF Description ID about parseType resource li nodeID datatype
```

Class names

```
Seq Bag Alt Statement Property XMLLiteral List
```

Property names

`subject predicate object type value first rest _`*n*
where *n* is a decimal integer greater than zero with no leading zeros.

Resource names

```
nil
```

Any other names are not defined and SHOULD generate a warning when encountered, but should otherwise behave normally.

Within RDF/XML documents it is not permitted to use XML namespaces whose namespace name is the ·RDF namespace URI reference· concatenated with additional characters.

Throughout this document the terminology `rdf:`*name* will be used to indicate *name* is from the RDF vocabulary and it has a RDF URI reference of the concatenation of the ·RDF namespace URI reference· and *name*. For example, `rdf:type` has the RDF URI reference `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

5.2 Identifiers

The RDF Graph (RDF Concepts and Abstract Syntax Section 3) defines three types of nodes and one type of predicate:

RDF URI reference nodes and predicates

RDF URI references (RDF Concepts and Abstract Syntax Section 3.1) can be either:

- given as XML attribute values interpreted as relative URI references that are resolved against the in-scope base URI as described in section 5.3 to give absolute RDF URI references
- transformed from XML namespace-qualified element and attribute names (QNames)
- transformed from `rdf:ID` attribute values.

Within RDF/XML, XML QNames are transformed into RDF URI references by appending the XML local name to the namespace name (URI reference). For example, if the XML namespace prefix `foo` has namespace name (URI reference) `http://example.org/somewhere/` then the QName `foo:bar` would correspond to the RDF URI reference `http://example.org/somewhere/bar`. Note that this restricts which RDF URI references can be made and the same URI can be given in multiple ways.

The `rdf:ID` values are transformed into RDF URI references by appending the attribute value to the result of appending "#" to the in-scope base URI which is defined in Section 5.3 Resolving URIs

Literal nodes (always object nodes)

RDF literals (RDF Concepts and Abstract Syntax 6.5) are either plain literals (ibid), or typed literals (ibid). The latter includes XML literals (ibid section 5, *XML Content within an RDF graph*).

Blank Node Identifiers

Blank nodes have distinct identity in the RDF graph. When the graph is written in a syntax such as RDF/XML, these blank nodes may need graph-local identifiers and a syntax in order to preserve this distinction. These local identifiers are called blank node identifiers and are used in RDF/XML as values of the `rdf:nodeID` attribute with the syntax given in Production nodeIdAttr. Blank node identifiers in RDF/XML are scoped to the XML Information Set document information item.

If no blank node identifier is given explicitly as an `rdf:nodeID` attribute value then one will need to be generated (using generated-blank-node-id, see section 6.3.3). Such generated blank node identifiers must not clash with any blank node identifiers derived from `rdf:nodeID` attribute values. This can be implemented by any method that preserves the distinct identity of all the blank nodes in the graph, that is, the same blank node identifier is not given for different blank nodes. One possible method would be to add a constant prefix to all the `rdf:nodeID` attribute values and ensure no generated

blank node identifiers ever used that prefix. Another would be to map all `rdf:nodeID` attribute values to new generated blank node identifiers and perform that mapping on all such values in the RDF/XML document.

5.3 Resolving URIs

RDF/XML supports XML Base [XML-BASE] which defines a ·base-uri· accessor for each ·root event· and ·element event·. Relative URI references are resolved into RDF URI references according to the algorithm specified in XML Base [XML-BASE] (and RFC 2396). These specifications do not specify an algorithm for resolving a fragment identifier alone, such as `#foo`, or the empty string `""` into an RDF URI reference. In RDF/XML, a fragment identifier is transformed into a RDF URI reference by appending the fragment identifier to the in-scope base URI. The empty string is transformed into an RDF URI reference by substituting the in-scope base URI.

Test: Indicated by test001.rdf and test001.nt

Test: Indicated by test004.rdf and test004.nt

Test: Indicated by test008.rdf and test008.nt

Test: Indicated by test013.rdf and test013.nt

Test: Indicated by test016.rdf and test016.nt

An empty same document reference "" resolves against the URI part of the base URI; any fragment part is ignored. See Uniform Resource Identifiers (URI) [URIS] section 4.2

Test: Indicated by test013.rdf and test013.nt

Implementation Note (Informative): When using a hierarchical base URI that has no path component (/), it must be added before using as a base URI for resolving.

Test: Indicated by test011.rdf and test011.nt

5.4 Constraints

constraint-id

Each application of production idAttr matches an attribute. The pair formed by the ·string-value· accessor of the matched attribute and the ·base-uri· accessor of the matched attribute is unique within a single RDF/XML document.

The syntax of the names must match the rdf-id production.

Test: Indicated by test014.rdf and test014.nt

5.5 Conformance

Definition:

An RDF Document is a serialization of an RDF Graph into a concrete syntax.

Definition:

An RDF/XML Document is an RDF Document written in the recommended XML transfer syntax for RDF as defined in this document.

Conformance:

An RDF/XML Document is a conforming RDF/XML document if it adheres to the specification defined in this document.

6 Syntax Data Model

This document specifies the syntax of RDF/XML as a grammar on an alphabet of symbols. The symbols are called *events* in the style of the [XPATH] Information Set Mapping. A sequence of events is normally derived from an XML document, in which case they are in document order as defined below in Section 6.2 Information Set Mapping. The sequence these events form are intended to be similar to the sequence of events produced by the [SAX2] XML API from the same XML document. Sequences of events may be checked against the grammar to determine whether they are or are not syntactically well-formed RDF/XML.

The grammar productions may include actions which fire when the production is recognized. Taken together these actions define a transformation from any syntactically well-formed RDF/XML sequence of events into an RDF graph represented in the N-Triples language.

The model given here illustrates one way to create a representation of an RDF Graph from an RDF/XML document. It does not mandate any implementation method — any other method that results in a representation of the same RDF Graph may be used.

In particular:

- This specification permits any representation of an RDF graph (see [RDF-CONCEPTS]); in particular, it does not require the use of N-Triples.
- This specification does not require the use of [XPATH] or [SAX2]
- This specification places no constraints on the order in which software transforming RDF/XML into a representation of a graph, constructs the representation of the graph.
- Software transforming RDF/XML into a representation of a graph MAY eliminate duplicate predicate arcs.

The syntax does not support non-well-formed XML documents, nor documents that otherwise do not have an XML Information Set; for example, that do not conform to Namespaces in XML [XML-NS].

The Infoset requires support for XML Base [XML-BASE]. RDF/XML uses the information item property [base URI], discussed in section 5.3

This specification requires an XML Information Set [INFOSET] which supports at least the following information items and properties for RDF/XML:

document information item
[document element], [children], [base URI]
element information item
[local name], [namespace name], [children], [attributes], [parent], [base URI]
attribute information item
[local name], [namespace name], [normalized value]
character information item
[character code]

There is no mapping of the following items to data model events:

processing instruction information item
unexpanded entity reference information item
comment information item
document type declaration information item
unparsed entity information item
notation information item
namespace information item

Other information items and properties have no mapping to syntax data model events.

Element information items with reserved XML Names (See Name in XML 1.0) are not mapped to data model element events. These are all those with property [prefix] beginning with `xml` (case independent comparison) and all those with [prefix] property having no value and which have [local name] beginning with `xml` (case independent comparison).

All information items contained inside XML elements matching the parseTypeLiteralPropertyElt production form XML literals and do not follow this mapping. See parseTypeLiteralPropertyElt for further information.

This section is intended to satisfy the requirements for Conformance in the [INFOSET] specification. It specifies the information items and properties that are needed to implement this specification.

6.1 Events

There are nine types of event defined in the following subsections. Most events are constructed from an Infoset information item (except for URI reference, blank node, plain literal and typed literal). The effect of an event constructor is to create a new event with a unique identity, distinct from all other events. Events have accessor operations on them and most have the *string-value* accessor that may be a static value or computed.

6.1.1 Root Event

Constructed from a document information item and takes the following accessors and values.

document-element
: Set to the value of document information item property [document-element].

children
: Set to the value of document information item property [children].

base-uri
: Set to the value of document information item property [base URI].

language
: Set to the empty string.

6.1.2 Element Event

Constructed from an element information item and takes the following accessors and values:

local-name
Set to the value of element information item property [local name].
namespace-name
Set to the value of element information item property [namespace name].
children
Set to the value of element information item property [children].
base-uri
Set to the value of element information item property [base URI].
attributes

Made from the value of element information item property [attributes] which is a set of attribute information items.

If this set contains an attribute information item `xml:lang` ([namespace name] property with the value "http://www.w3.org/XML/1998/namespace" and [local name] property value "lang") it is removed from the set of attribute information items and the ·language· accessor is set to the [normalized-value] property of the attribute information item.

All remaining reserved XML Names (See Name in XML 1.0) are now removed from the set. These are, all attribute information items in the set with property [prefix] beginning with `xml` (case independent comparison) and all attribute information items with [prefix] property having no value and which have [local name] beginning with `xml` (case independent comparison) are removed. Note that the [base URI] accessor is computed by XML Base before any `xml:base` attribute information item is deleted.

The remaining set of attribute information items are then used to construct a new set of Attribute Events which is assigned as the value of this accessor.

URI
Set to the string value of the concatenation of the value of the namespace-name accessor and the value of the local-name accessor.
URI-string-value

The value is the concatenation of the following in this order "<", the escaped value of the ·URI· accessor and ">".

The escaping of the ·URI· accessor uses the N-Triples escapes for URI references as described in 3.3 URI References.

li-counter

Set to the integer value 1.

language

Set from the ·attributes· as described above. If no value is given from the attributes, the value is set to the value of the language accessor on the parent event (either a Root Event or an Element Event), which may be the empty string.

subject

Has no initial value. Takes a value that is an Identifier event. This accessor is used on elements that deal with one node in the RDF graph, this generally being the subject of a statement.

6.1.3 End Element Event

Has no accessors. Marks the end of the containing element in the sequence.

6.1.4 Attribute Event

Constructed from an attribute information item and takes the following accessors and values:

local-name

Set to the value of attribute information item property [local name].

namespace-name

Set to the value of attribute information item property [namespace name].

string-value

Set to the value of the attribute information item property [normalized value] as specified by [XML] (if an attribute whose normalized value is a zero-length string, then the string-value is also a zero-length string).

URI

If ·namespace-name· is present, set to a string value of the concatenation of the value of the ·namespace-name· accessor and the value of the ·local-name· accessor. Otherwise if ·local-name· is `ID`, `about`, `resource`, `parseType` or `type`, set to a string value of the concatenation of the ·RDF namespace URI reference· and the value of the ·local-name· accessor. Other non-namespaced ·local-name· accessor values are forbidden.

The support for a limited set of non-namespaced names is REQUIRED and intended to allow RDF/XML documents specified in [RDF-MS] to remain valid; new documents SHOULD NOT use these unqualified attributes and applications MAY choose to warn when the unqualified form is seen in a document.

The construction of RDF URI references from XML attributes can generate the same RDF URI references from different

XML attributes. This can cause ambiguity in the grammar when matching attribute events (such as when `rdf:about` and `about` XML attributes are both present). Documents that have this are illegal.

URI-string-value

The value is the concatenation of the following in this order "<", the escaped value of the ·URI· accessor and ">".

The escaping of the ·URI· accessor uses the N-Triples escapes for URI references as described in 3.3 URI References.

6.1.5 Text Event

Constructed from a sequence of one or more consecutive character information items. Has the single accessor:

string-value
Set to the value of the string made from concatenating the [character code] property of each of the character information items.

6.1.6 URI Reference Event

An event for a RDF URI references which has the following accessors:

identifier
Takes a string value used as an RDF URI reference.

string-value

The value is the concatenation of "<", the escaped value of the ·identifier· accessor and ">"

The escaping of the ·identifier· accessor value uses the N-Triples escapes for URI references as described in 3.3 URI References.

These events are constructed by giving a value for the ·identifier· accessor.

For further information on identifiers in the RDF graph, see section 5.2.

6.1.7 Blank Node Identifier Event

An event for a blank node identifier which has the following accessors:

identifier
: Takes a string value.

string-value
: The value is a function of the value of the ·identifier· accessor. The value begins with "_:" and the entire value MUST match the N-Triples nodeID production. The function MUST preserve distinct blank node identity as discussed in in section 5.2 Identifiers.

These events are constructed by giving a value for the ·identifier· accessor.

For further information on identifiers in the RDF graph, see section 5.2.

6.1.8 Plain Literal Event

An event for a plain literal which can have the following accessors:

literal-value
: Takes a string value.

literal-language
: Takes a string value used as a language tag in an RDF plain literal.

string-value

: The value is calculated from the other accessors as follows.

: If ·literal-language· is the empty string then the value is the concatenation of """ (1 double quote), the escaped value of the ·literal-value· accessor and """ (1 double quote).

: Otherwise the value is the concatenation of """ (1 double quote), the escaped value of the ·literal-value· accessor ""@" (1 double quote and a '@'), and the value of the ·literal-language· accessor.

: The escaping of the ·literal-value· accessor value uses the N-Triples escapes for strings as described in 3.2 Strings for escaping certain characters such as ".

These events are constructed by giving values for the ·literal-value· and ·literal-language· accessors.

Interoperability Note (Informative): Literals beginning with a Unicode combining character are allowed however they may cause interoperability problems. See [CHARMOD] for further information.

6.1.9 Typed Literal Event

An event for a typed literal which can have the following accessors:

literal-value
: Takes a string value.

literal-datatype
: Takes a string value used as an RDF URI reference.

string-value

The value is the concatenation of the following in this order """ (1 double quote), the escaped value of the ·literal-value· accessor, """ (1 double quote), "^^<", the escaped value of the ·literal-datatype· accessor and ">".

The escaping of the ·literal-value· accessor value uses the N-Triples escapes for strings as described in 3.2 Strings for escaping certain characters such as ". The escaping of the ·literal-datatype· accessor value must use the N-Triples escapes for URI references as described in 3.3 URI References.

These events are constructed by giving values for the ·literal-value· and ·literal-datatype· accessors.

Interoperability Note (Informative): Literals beginning with a Unicode combining character are allowed however they may cause interoperability problems. See [CHARMOD] for further information.

Implementation Note (Informative): In XML Schema (part 1) [XML-SCHEMA1], white space normalization occurs during validation according to the value of the whiteSpace facet. The syntax mapping used in this document occurs after this, so the whiteSpace facet formally has no further effect.

6.2 Information Set Mapping

To transform the Infoset into the sequence of events in *document order*, each information item is transformed as described above to generate a tree of events with accessors and values. Each element event is then replaced as described below to turn the tree of events into a sequence in document order.

1. The original element event
2. The value of the children accessor recursively transformed, a possibly empty ordered list of events.
3. An end element event

6.3 Grammar Notation

The following notation is used to describe matching the sequence of data model events as given in Section 6 and the actions to perform for the matches. The RDF/XML grammar is defined in terms of mapping from these matched data model events to triples, using notation of the form:

```
number event-type event-content
action...
N-Triples
```

where the *event-content* is an expression matching *event-types* (as defined in Section 6.1), using notation given in the following sections. The number is used for reference purposes. The grammar *action* may include generating new triples to the graph, written in N-Triples format.

The following sections describe the general notation used and that for event matching and actions.

6.3.1 Grammar General Notation

Grammar General Notation.

Notation	Meaning
event.accessor	The value of an event accessor.
`rdf:`*X*	A URI as defined in section 5.1.
"ABC"	A string of characters A, B, C in order.

6.3.2 Grammar Event Matching Notation

Grammar Event Matching Notation.

Notation	Meaning
A == B	Event accessor A matches expression B.
A != B	A is not equal to B.
A | B | ...	The A, B, ... terms are alternatives.
A - B	The terms in A excluding all the terms in B.
anyURI.	Any URI.
anyString.	Any string.
list(item1, item2, ...); list()	An ordered list of events. An empty list.
set(item1, item2, ...); set()	An unordered set of events. An empty set.
*	Zero or more of preceding term.
?	Zero or one of preceding term.
+	One or more of preceding term.
root(acc1 == value1, acc2 == value2, ...)	Match a Root Event with accessors.
start-element(acc1 == value1, acc2 == value2, ...) *children* end-element()	Match a sequence of Element Event with accessors, a possibly empty list of events as element content and an End Element Event.
attribute(acc1 == value1, acc2 == value2, ...)	Match an Attribute Event with accessors.
text()	Match a Text Event.

6.3.3 Grammar Action Notation

Grammar Action Notation.

Notation	Meaning
A := B	Assigns A the value B.
concat(A, B, ..)	A string created by concatenating the terms in order.
resolve(*e*, *s*)	A string created by interpreting string *s* as a relative URI reference to the ·base-uri· accessor of *e* as defined in Section 5.3 Resolving URIs. The resulting string represents an RDF URI reference.
generated-blank-node-id()	A string value for a new distinct generated blank node identifier as defined in section 5.2 Identifiers.
event.accessor := *value*	Sets an event accessor to the given value.
uri(identifier := value)	Create a new URI Reference Event.
bnodeid(identifier := value)	Create a new Blank Node Identifier Event. See also section 5.2 Identifiers.
literal(literal-value := string, literal-language := language, ...)	Create a new Plain Literal Event.
typed-literal(literal-value := string, ...)	Create a new Typed Literal Event.

7 RDF/XML Grammar

7.1 Grammar summary

7.2.2 coreSyntaxTerms	`rdf:RDF` | `rdf:ID` | `rdf:about` | `rdf:parseType` | `rdf:resource` | `rdf:nodeID` | `rdf:datatype`
7.2.3 syntaxTerms	coreSyntaxTerms | `rdf:Description` | `rdf:li`
7.2.4 oldTerms	`rdf:aboutEach` | `rdf:aboutEachPrefix` | `rdf:bagID`
7.2.5 nodeElementURIs	anyURI - (coreSyntaxTerms | `rdf:li` | oldTerms)
7.2.6 propertyElementURIs	anyURI - (coreSyntaxTerms | `rdf:Description` | oldTerms)

7.2.7 propertyAttributeURIs	anyURI - (coreSyntaxTerms | `rdf:Description` | `rdf:li` | oldTerms)
7.2.8 doc	root(document-element == RDF, children == list(RDF))
7.2.9 RDF	start-element(URI == `rdf:RDF`, attributes == set()) nodeElementList end-element()
7.2.10 nodeElementList	ws* (nodeElement ws*)*
7.2.11 nodeElement	start-element(URI == nodeElementURIs attributes == set((idAttr | nodeIdAttr | aboutAttr)?, propertyAttr*)) propertyEltList end-element()
7.2.12 ws	A text event matching white space defined by [XML] definition *White Space* Rule [3] S in section Common Syntactic Constructs
7.2.13 propertyEltList	ws* (propertyElt ws*) *
7.2.14 propertyElt	resourcePropertyElt | literalPropertyElt | parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt | parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt | emptyPropertyElt
7.2.15 resourcePropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?)) ws* nodeElement ws* end-element()
7.2.16 literalPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, datatypeAttr?)) text() end-element()
7.2.17 parseTypeLiteralPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseLiteral)) literal end-element()
7.2.18 parseTypeResourcePropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseResource)) propertyEltList end-element()
7.2.19 parseTypeCollectionPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseCollection)) nodeElementList end-element()
7.2.20	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseOther))

parseTypeOtherPropertyElt	propertyEltList end-element()
7.2.21 emptyPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, (resourceAttr | nodeIdAttr)?, propertyAttr*)) end-element()
7.2.22 idAttr	attribute(URI == `rdf:ID`, string-value == rdf-id)
7.2.23 nodeIdAttr	attribute(URI == `rdf:nodeID`, string-value == rdf-id)
7.2.24 aboutAttr	attribute(URI == `rdf:about`, string-value == URI-reference)
7.2.25 propertyAttr	attribute(URI == propertyAttributeURIs, string-value == anyString)
7.2.26 resourceAttr	attribute(URI == `rdf:resource`, string-value == URI-reference)
7.2.27 datatypeAttr	attribute(URI == `rdf:datatype`, string-value == URI-reference)
7.2.28 parseLiteral	attribute(URI == `rdf:parseType`, string-value == "Literal")
7.2.29 parseResource	attribute(URI == `rdf:parseType`, string-value == "Resource")
7.2.30 parseCollection	attribute(URI == `rdf:parseType`, string-value == "Collection")
7.2.31 parseOther	attribute(URI == `rdf:parseType`, string-value == anyString - ("Resource" | "Literal" | "Collection"))
7.2.32 URI-reference	An RDF URI reference.
7.2.33 literal	Any XML element content that is allowed according to [XML] definition *Content of Elements* Rule [43] content. in section 3.1 Start-Tags, End-Tags, and Empty-Element Tags
7.2.34 rdf-id	An attribute ·string-value· matching any legal [XML-NS] token NCName

7.2 Grammar Productions

7.2.1 Grammar start

If the RDF/XML is a standalone XML document (identified by presentation as an application/rdf+xml RDF MIME type object, or by some other means) then the grammar may start with production doc or production nodeElement.

If the content is known to be RDF/XML by context, such as when RDF/XML is embedded inside other XML content, then the grammar can either start at Element Event RDF (only when an element is legal at that point in the XML) or at production

nodeElementList (only when element content is legal, since this is a list of elements). For such embedded RDF/XML, the ·base-uri· value on the outermost element must be initialized from the containing XML since no Root Event will be available. Note that if such embedding occurs, the grammar may be entered several times but no state is expected to be preserved.

7.2.2 Production coreSyntaxTerms

```
rdf:RDF | rdf:ID | rdf:about | rdf:parseType | rdf:resource | rdf:nodeID | rdf:datatype
```

A subset of the syntax terms from the RDF vocabulary in section 5.1 which are used in RDF/XML.

7.2.3 Production syntaxTerms

```
coreSyntaxTerms | rdf:Description | rdf:li
```

All the syntax terms from the RDF vocabulary in section 5.1 which are used in RDF/XML.

7.2.4 Production oldTerms

```
rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID
```

These are the names from the RDF vocabulary that have been withdrawn from the language. See the resolutions of Issue rdfms-aboutEach-on-object, Issue rdfms-abouteachprefix and Last Call Issue timbl-01 for further information.

Error Test: Indicated by error001.rdf and error002.rdf

7.2.5 Production nodeElementURIs

```
anyURI - ( coreSyntaxTerms | rdf:li | oldTerms )
```

The RDF URI references that are allowed on node elements.

7.2.6 Production propertyElementURIs

anyURI - (coreSyntaxTerms | `rdf:Description` | oldTerms)

The URIs that are allowed on property elements.

7.2.7 Production propertyAttributeURIs

anyURI - (coreSyntaxTerms | `rdf:Description` | `rdf:li` | oldTerms)

The RDF URI references that are allowed on property attributes.

7.2.8 Production doc

root(document-element == RDF,
  children == list(RDF))

7.2.9 Production RDF

start-element(URI == `rdf:RDF`,
  attributes == set())
nodeElementList
end-element()

7.2.10 Production nodeElementList

ws* (nodeElement ws*)*

7.2.11 Production nodeElement

start-element(URI == nodeElementURIs
  attributes == set((idAttr | nodeIdAttr | aboutAttr)?, propertyAttr*))
propertyEltList
end-element()

For node element *e*, the processing of some of the attributes has to be done before other work such as dealing with children events or other attributes. These can be processed in any order:

- If there is an attribute *a* with *a*.URI == `rdf:ID`, then *e*.subject := uri(identifier := resolve(*e*, concat("#", *a*.string-value))).
- If there is an attribute *a* with *a*.URI == `rdf:nodeID`, then *e*.subject := bnodeid(identifier:=*a*.string-value).
- If there is an attribute *a* with *a*.URI == `rdf:about` then *e*.subject := uri(identifier := resolve(*e*, *a*.string-value)).

If *e*.subject is empty, then *e*.subject := bnodeid(identifier := generated-blank-node-id()).

The following can then be performed in any order:

- If *e*.URI != `rdf:Description` then the following statement is added to the graph:

```
e.subject.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> e.URI-string-value .
```

- If there is an attribute *a* in propertyAttr with *a*.URI == `rdf:type` then *u*:=uri(identifier:=resolve(*a*.string-value)) and the following tiple is added to the graph:

```
e.subject.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> u.string-value .
```

- For each attribute *a* matching propertyAttr (and not `rdf:type`), the Unicode string *a*.string-value SHOULD be in Normal Form C[NFC], *o* := literal(literal-value := *a*.string-value, literal-language := *e*.language) and the following statement is added to the graph:

```
e.subject.string-value a.URI-string-value o.string-value .
```

- Handle the propertyEltList children events in document order.

7.2.12 Production ws

A text event matching white space defined by [XML] definition *White Space* Rule [3] S in section Common Syntactic Constructs

7.2.13 Production propertyEltList

ws* (propertyElt ws*) *

7.2.14 Production propertyElt

resourcePropertyElt | literalPropertyElt | parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt | parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt | emptyPropertyElt

If element *e* has *e*.URI = `rdf:li` then apply the list expansion rules on element *e*.parent in section 7.4 to give a new URI *u* and *e*.URI := *u*.

The action of this production must be done before the actions of any sub-matches (resourcePropertyElt ... emptyPropertyElt). Alternatively the result must be equivalent to as if it this action was performed first, such as performing as the first action of all of the sub-matches.

7.2.15 Production resourcePropertyElt

start-element(URI == propertyElementURIs),
  attributes == set(idAttr?))
ws* nodeElement ws*
end-element()

For element *e*, and the single contained nodeElement *n*, first *n* must be processed using production nodeElement. Then the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value n.subject.string-value .
```

If the `rdf:ID` attribute *a* is given, the above statement is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*

7.2.16 Production literalPropertyElt

```
start-element(URI == propertyElementURIs ),
   attributes == set(idAttr?, datatypeAttr?))
text()
end-element()
```

Note that the empty literal case is defined in production emptyPropertyElt.

For element *e*, and the text event *t*. The Unicode string *t*.string-value SHOULD be in Normal Form C[NFC]. If the `rdf:datatype` attribute *d* is given then *o* := typed-literal(literal-value := *t*.string-value, literal-datatype := *d*.string-value) otherwise *o* := literal(literal-value := *t*.string-value, literal-language := *e*.language) and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value o.string-value .
```

If the `rdf:ID` attribute *a* is given, the above statement is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*.

7.2.17 Production parseTypeLiteralPropertyElt

```
start-element(URI == propertyElementURIs ),
   attributes == set(idAttr?, parseLiteral))
literal
end-element()
```

For element *e* and the literal *l* that is the `rdf:parseType="Literal"` content. *l* is not transformed by the syntax data model mapping into events (as noted in 6 Syntax Data Model) but remains an XML Infoset of XML Information items.

l is transformed into the lexical form of an XML literal in the RDF graph *x* (a Unicode string) by the following algorithm. This does not mandate any implementation method — any other method that gives the same result may be used.

1. Use *l* to construct an XPath[XPATH] node-set (a document subset)
2. Apply Exclusive XML Canonicalization [XML-XC14N]) with comments and with empty InclusiveNamespaces PrefixList

to this node-set to give a sequence of octets *s*
3. This sequence of octets *s* can be considered to be a UTF-8 encoding of some Unicode string *x* (sequence of Unicode characters)
4. The Unicode string *x* is used as the lexical form of *l*
5. This Unicode string *x* SHOULD be in NFC Normal Form C[NFC]

Then *o* := typed-literal(literal-value := *x*, literal-datatype := `http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral`) and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value o.string-value .
```

Test: Empty literal case indicated by test009.rdf and test009.nt

If the `rdf:ID` attribute *a* is given, the above statement is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*.

7.2.18 Production parseTypeResourcePropertyElt

```
start-element(URI == propertyElementURIs ),
   attributes == set(idAttr?, parseResource))
propertyEltList
end-element()
```

For element *e* with possibly empty element content *c*.

n := bnodeid(identifier := generated-blank-node-id()).

Add the following statement to the graph:

```
e.parent.subject.string-value e.URI-string-value n.string-value .
```

Test: Indicated by test004.rdf and test004.nt

If the `rdf:ID` attribute *a* is given, the statement above is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*.

If the element content *c* is not empty, then use event *n* to create a new sequence of events as follows:

```
start-element(URI := rdf:Description,
   subject := n,
   attributes := set())
c
end-element()
```

Then process the resulting sequence using production nodeElement.

7.2.19 Production parseTypeCollectionPropertyElt

```
start-element(URI == propertyElementURIs ),
   attributes == set(idAttr?, parseCollection))
nodeElementList
end-element()
```

For element event *e* with possibly empty nodeElementList *l*. Set *s*:=list().

For each element event *f* in *l*, *n* := bnodeid(identifier := generated-blank-node-id()) and append *n* to *s* to give a sequence of events.

If *s* is not empty, *n* is the first event identifier in *s* and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value n.string-value .
```

otherwise the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
```

If the `rdf:ID` attribute *a* is given, either of the the above statements is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3.

If *s* is empty, no further work is performed.

For each event *n* in *s* and the corresponding element event *f* in *l*, the following statement is added to the graph:

```
n.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> f.string-value .
```

For each consecutive and overlapping pair of events (*n*, *o*) in *s*, the following statement is added to the graph:

```
n.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> o.string-value .
```

If *s* is not empty, *n* is the last event identifier in *s*, the following statement is added to the graph:

```
n.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
```

7.2.20 Production parseTypeOtherPropertyElt

```
start-element(URI == propertyElementURIs ),
   attributes == set(idAttr?, parseOther))
propertyEltList
end-element()
```

All `rdf:parseType` attribute values other than the strings "Resource", "Literal" or "Collection" are treated as if the value was "Literal". This production matches and acts as if production parseTypeLiteralPropertyElt was matched. No extra triples are generated for other `rdf:parseType` values.

7.2.21 Production emptyPropertyElt

```
start-element(URI == propertyElementURIs ),
```

attributes == set(idAttr?, (resourceAttr | nodeIdAttr)?, propertyAttr*))
end-element()

- If there are no attributes **or** only the optional `rdf:ID` attribute *i* then *o* := literal(literal-value:="", literal-language := *e*.language) and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value o.string-value .
```

  and then if *i* is given, the above statement is reified with uri(identifier := resolve(*e*, concat("#", *i*.string-value))) using the reification rules in section 7.3.

  Test: Indicated by test002.rdf and test002.nt

  Test: Indicated by test005.rdf and test005.nt

- Otherwise
  - If `rdf:resource` attribute *i* is present, then *r* := uri(identifier := resolve(*e*, *i*.string-value))
  - If `rdf:nodeID` attribute *i* is present, then *r* := bnodeid(identifier := *i*.string-value)
  - If neither, *r* := bnodeid(identifier := generated-blank-node-id())

  The following are done in any order:

  - For all propertyAttr attributes *a* (in any order)
    - If *a*.URI == `rdf:type` then *u*:=uri(identifier:=resolve(*a*.string-value)) and the following triple is added to the graph:

```
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> u.string-value .
```

    - Otherwise Unicode string *a*.string-value SHOULD be in Normal Form C[NFC], *o* := literal(literal-value := *a*.string-value, literal-language := *e*.language) and the following statement is added to the graph:

```
r.string-value a.URI-string-value o.string-value .
```

Test: Indicated by test013.rdf and test013.nt

Test: Indicated by test014.rdf and test014.nt

- Add the following statement to the graph:

```
e.parent.subject.string-value e.URI-string-value r.string-value .
```

and then if `rdf:ID` attribute *i* is given, the above statement is reified with uri(identifier := resolve(*e*, concat("#", *i*.string-value))) using the reification rules in section 7.3.

7.2.22 Production idAttr

```
attribute(URI == rdf:ID,
   string-value == rdf-id)
```

Constraint:: constraint-id applies to the values of `rdf:ID` attributes

7.2.23 Production nodeIdAttr

```
attribute(URI == rdf:nodeID,
   string-value == rdf-id)
```

7.2.24 Production aboutAttr

```
attribute(URI == rdf:about,
   string-value == URI-reference)
```

7.2.25 Production propertyAttr

attribute(URI == propertyAttributeURIs,
  string-value == anyString)

7.2.26 Production resourceAttr

attribute(URI == `rdf:resource`,
  string-value == URI-reference)

7.2.27 Production datatypeAttr

attribute(URI == `rdf:datatype`,
  string-value == URI-reference)

7.2.28 Production parseLiteral

attribute(URI == `rdf:parseType`,
  string-value == "Literal")

7.2.29 Production parseResource

attribute(URI == `rdf:parseType`,
  string-value == "Resource")

7.2.30 Production parseCollection

attribute(URI == `rdf:parseType`,
  string-value == "Collection")

7.2.31 Production parseOther

attribute(URI == `rdf:parseType`,
  string-value == anyString - ("Resource" | "Literal" | "Collection"))

7.2.32 Production URI-reference

An RDF URI reference.

7.2.33 Production literal

Any XML element content that is allowed according to [XML] definition *Content of Elements* Rule [43] content. in section 3.1 Start-Tags, End-Tags, and Empty-Element Tags
The string-value for the resulting event is discussed in section 7.2.17.

7.2.34 Production rdf-id

An attribute ·string-value· matching any legal [XML-NS] token NCName

7.3 Reification Rules

For the given URI reference event *r* and the statement with terms *s*, *p* and *o* corresponding to the N-Triples:

```
s p o .
```

add the following statements to the graph:

```
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#subject> s .
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate> p .
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#object> o .
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-
```

```
ns#Statement> .
```

7.4 List Expansion Rules

For the given element *e*, create a new RDF URI reference *u* := concat("http://www.w3.org/1999/02/22-rdf-syntax-ns#_", *e*.li-counter), increment the *e*.li-counter property by 1 and return *u*.

8 Serializing an RDF Graph to RDF/XML

There are some RDF Graphs as defined in RDF Concepts and Abstract Syntax that cannot be serialized in RDF/XML. These are those that:

Use property names that cannot be turned into XML namespace-qualified names.
: An XML namespace-qualified name (QName) has restrictions on the legal characters such that not all property URIs can be expressed as these names. It is recommended that implementors of RDF serializers, in order to break a URI into a namespace name and a local name, split it after the last XML non-NCName character, ensuring that the first character of the name is a Letter or '_'. If the URI ends in a non-NCName character then throw a "this graph cannot be serialized in RDF/XML" exception or error.

Use inappropriate reserved names as properties
: For example, a property with the same URI as any of the syntaxTerms production.

Implementation Note (Informative): When an RDF graph is serialized to RDF/XML and has an XML Schema Datatype (XSD), it SHOULD be written in a form that does not require whitespace processing. XSD support is NOT required by RDF or RDF/XML so this is optional.

9 Using RDF/XML with HTML and XHTML

If RDF/XML is embedded inside HTML or XHTML this can add many new elements and attributes, many of which will not be in the appropriate DTD. This embedding causes validation against the DTD to fail. The obvious solution of changing or extending the DTD is not practical for most uses. This problem has been analyzed extensively by Sean B. Palmer in RDF in HTML: Approaches [RDF-IN-XHTML] and it concludes that there is no single embedding method that satisfies all applications and remains simple.

The recommended approach is to not embed RDF/XML in HTML/XHTML but rather to use `<link>` element in the `<head>`

element of the HTML/HTML to point at a separate RDF/XML document. This approach has been used for several years by the Dublin Core Metadata Initiative (DCMI) on its Web site.

To use this technique, the `<link>` element `href` should point at the URI of the RDF/XML content and the `type` attribute should be used with the value of `"application/rdf+xml"`, the proposed MIME type for RDF/XML, see Section 4

The value of the `rel` attribute may also be set to indicate the relationship; this is an application dependent value. The DCMI has used and recommended `rel="meta"` when linking in RFC 2731 — Encoding Dublin Core Metadata in HTML[RFC-2731] however `rel="alternate"` may also be appropriate. See HTML 4.01 link types, XHTML Modularization — LinkTypes and XHTML 2.0 — LinkTypes for further information on the values that may be appropriate for the different versions of HTML.

Example 21 shows using this method with the `link` tag inside an XHTML document to link to an external RDF/XML document.

Example 21: Using `link` in XHTML with an external RDF/XML document (example21.html linking to example21.rdf)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>My document</title>
    <meta http-equiv="Content-type" content='text/html; charset="utf-8"' />
    <link rel="alternate" type="application/rdf+xml" title="RDF Version" href="example21.rdf" />
  </head>
  <body>
    <h1>My document</h1>

  </body>
</html>
```

10 Using RDF/XML with SVG (Informative)

There is a standardized approach for associating RDF compatible metadata with SVG — the metadata element which was explicitly designed for this purpose as defined in Section 21 Metadata of the Scalable Vector Graphics (SVG) 1.0 Specification [SVG] and Section 21 Metadata of the Scalable Vector Graphics (SVG) 1.1 Specification [SVG11].

This document contains two example graphs in SVG with such embedded RDF/XML inside the metadata element: figure 1

and figure 2.

11 Acknowledgments (Informative)

The following people provided valuable contributions to the document:

- Dan Brickley, W3C/ILRT
- Jeremy Carroll, HP Labs, Bristol
- Graham Klyne, Nine by Nine
- Bijan Parsia, MIND Lab at University of Maryland at College Park

This document is a product of extended deliberations by the RDF Core working group, whose members have included: Art Barstow (W3C) Dave Beckett (ILRT), Dan Brickley (W3C/ILRT), Dan Connolly (W3C), Jeremy Carroll (Hewlett Packard), Ron Daniel (Interwoven Inc), Bill dehOra (InterX), Jos De Roo (AGFA), Jan Grant (ILRT), Graham Klyne (Clearswift and Nine by Nine), Frank Manola (MITRE Corporation), Brian McBride (Hewlett Packard), Eric Miller (W3C), Stephen Petschulat (IBM), Patrick Stickler (Nokia), Aaron Swartz (HWG), Mike Dean (BBN Technologies / Verizon), R. V. Guha (Alpiri Inc), Pat Hayes (IHMC), Sergey Melnik (Stanford University), Martyn Horner (Profium Ltd).

This specification also draws upon an earlier RDF Model and Syntax document edited by Ora Lassilla and Ralph Swick, and RDF Schema edited by Dan Brickley and R. V. Guha. RDF and RDF Schema Working group members who contributed to this earlier work are: Nick Arnett (Verity), Tim Berners-Lee (W3C), Tim Bray (Textuality), Dan Brickley (ILRT / University of Bristol), Walter Chang (Adobe), Sailesh Chutani (Oracle), Dan Connolly (W3C), Ron Daniel (DATAFUSION), Charles Frankston (Microsoft), Patrick Gannon (CommerceNet), RV Guha (Epinions, previously of Netscape Communications), Tom Hill (Apple Computer), Arthur van Hoff (Marimba), Renato Iannella (DSTC), Sandeep Jain (Oracle), Kevin Jones, (InterMind), Emiko Kezuka (Digital Vision Laboratories), Joe Lapp (webMethods Inc.), Ora Lassila (Nokia Research Center), Andrew Layman (Microsoft), Ralph LeVan (OCLC), John McCarthy (Lawrence Berkeley National Laboratory), Chris McConnell (Microsoft), Murray Maloney (Grif), Michael Mealling (Network Solutions), Norbert Mikula (DataChannel), Eric Miller (OCLC), Jim Miller (W3C, emeritus), Frank Olken (Lawrence Berkeley National Laboratory), Jean Paoli (Microsoft), Sri Raghavan (Digital/Compaq), Lisa Rein (webMethods Inc.), Paul Resnick (University of Michigan), Bill Roberts (KnowledgeCite), Tsuyoshi Sakata (Digital Vision Laboratories), Bob Schloss (IBM), Leon Shklar (Pencom Web Works), David Singer (IBM), Wei (William) Song (SISU), Neel Sundaresan (IBM), Ralph Swick (W3C), Naohiko Uramoto (IBM), Charles Wicksteed (Reuters Ltd.), Misha Wolf (Reuters Ltd.), Lauren Wood (SoftQuad).

12 References

Normative References

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

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

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

[INFOSET]
XML Information Set, J. Cowan and R. Tobin, Editors. World Wide Web Consortium. 24 October 2001. This version is http://www.w3.org/TR/2001/REC-xml-infoset-20011024. The latest version of XML Information set is available at http://www.w3.org/TR/xml-infoset.

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

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

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

[KEYWORDS]
RFC 2119 — Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, IETF. March 1997. This document is http://www.ietf.org/rfc/rfc2119.txt.

[RFC-3023]
RFC 3023 — XML Media Types, M. Murata, S. St.Laurent, D.Kohn, IETF. January 2001. This document is http://www.ietf.org/rfc/rfc3023.txt.

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

[XML-BASE]
XML Base, J. Marsh, Editor, W3C Recommendation. World Wide Web Consortium, 27 June 2001. This version of XML Base is http://www.w3.org/TR/2001/REC-xmlbase-20010627. The latest version of XML Base is at http://www.w3.org/TR/xmlbase.

[XML-XC14N]
Exclusive XML Canonicalization Version 1.0, J. Boyer, D.E. Eastlake 3rd, J. Reagle, Authors/Editors. W3C Recommendation. World Wide Web Consortium, 18 July 2002. This version of Exclusive XML Canonicalization is http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718. The latest version of Canonical XML is at http://www.w3.org/TR/xml-exc-c14n.

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

[NFC]
Unicode Normalization Forms, Unicode Standard Annex #15, Mark Davis, Martin Dürst. (See http://www.unicode.org/unicode/reports/tr15/ for the latest version).

Informational References

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

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

[RDF-PRIMER]
RDF Primer, F. Manola, E. Miller, Editors, W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. The latest version is at http://www.w3.org/TR/rdf-primer/.

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

[STRIPEDRDF]
RDF: Understanding the Striped RDF/XML Syntax, D. Brickley, W3C, 2001. This document is http://www.w3.org/2001/10/stripes/.

[SVG]
Scalable Vector Graphics (SVG) 1.0 Specification, J. Ferraiolo (editor), 4 September 2001, W3C Recommendation. This version of SVG is http://www.w3.org/TR/2001/REC-SVG-20010904. The latest version of SVG is at http://www.w3.org/TR/SVG.

[SVG11]
Scalable Vector Graphics (SVG) 1.1 Specification, J. Ferraiolo, J. FUJISAWA, D. Jackson (editors), 14 January 2003, W3C Recommendation. This version of SVG is http://www.w3.org/TR/2003/REC-SVG11-20030114/. The latest version of SVG is at http://www.w3.org/TR/SVG11.

[XPATH]
XML Path Language (XPath) Version 1.0, J. Clark and S. DeRose, Editors. World Wide Web Consortium, 16 November 1999. This version of XPath is http://www.w3.org/TR/1999/REC-xpath-19991116. The latest version of XPath is at http://www.w3.org/TR/xpath.

[SAX2]
SAX Simple API for XML, version 2, D. Megginson, SourceForge, 5 May 2000. This document is http://sax.sourceforge.net/.

[UNPARSING]
Unparsing RDF/XML, J. J. Carroll, HP Labs Technical Report, HPL-2001-294, 2001. This document is available at http://www.hpl.hp.com/techreports/2001/HPL-2001-294.html.

[RELAXNG]
RELAX NG Specification, James Clark and MURATA Makoto, Editors, OASIS Committee Specification, 3 December 2001. This version of RELAX NG is http://www.oasis-open.org/committees/relax-ng/spec-20011203.html. The latest version of the RELAX NG Specification is at http://www.oasis-open.org/committees/relax-ng/spec.html.

[RELAXNG-COMPACT]
RELAX NG Compact Syntax, James Clark, Editor. OASIS Committee Specification, 21 November 2002. This document is http://www.oasis-open.org/committees/relax-ng/compact-20021121.html.

[RDF-IN-XHTML]
RDF in HTML: Approaches, Sean B. Palmer, 2002

[RFC-2731]

RFC 2731 — Encoding Dublin Core Metadata in HTML, John Kunze, DCMI, December 1999.

[XML-SCHEMA1]

XML Schema Part 1: Structures, H.S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, Editors, World Wide Web Consortium Recommendation, 2 May 2001. This version of XML Schema Part 1: Structures is http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/. The latest version of XML Schema Part 1: Structures is at http://www.w3.org/TR/xmlschema-1.

[XML-SCHEMA2]

XML Schema Part 2: Datatypes, P.V. Biron, A. Malhotra, Editors, World Wide Web Consortium Recommendation, 2 May 2001. This version of XML Schema Part 2: Datatypes is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. The latest version of XML Schema Part 2: Datatypes is at http://www.w3.org/TR/xmlschema-2.

A Syntax Schemas (Informative)

This appendix contains XML schemas for validating RDF/XML forms. These are example schemas for information only and are not part of this specification.

A.1 RELAX NG Compact Schema (Informative)

This is an example schema in RELAX NG Compact (for ease of reading) for RDF/XML. Applications can also use the RELAX NG XML version. These formats are described in RELAX NG ([RELAXNG]) and RELAX NG Compact ([RELAXNG-COMPACT]).

> **Note:** The RNGC schema has been updated to attempt to match the grammar but this has not been checked or used to validate RDF/XML.

RELAX NG Compact Schema for RDF/XML

```
#
# RELAX NG Compact Schema for RDF/XML Syntax
#
# This schema is for information only and NON-NORMATIVE
#
# It is based on one originally written by James Clark in
# http://lists.w3.org/Archives/Public/www-rdf-comments/2001JulSep/0248.html
# and updated with later changes.
#
```

```
namespace local = ""
namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"


start = doc


# I cannot seem to do this in RNGC so they are expanded in-line

# coreSyntaxTerms = rdf:RDF | rdf:ID | rdf:about | rdf:parseType | rdf:resource | rdf:nodeID | rdf:datatype
# syntaxTerms = coreSyntaxTerms | rdf:Description | rdf:li
# oldTerms    = rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID
# nodeElementURIs       = * - ( coreSyntaxTerms | rdf:li | oldTerms )
# propertyElementURIs   = * - ( coreSyntaxTerms | rdf:Description | oldTerms )
# propertyAttributeURIs = * - ( coreSyntaxTerms | rdf:Description | rdf:li | oldTerms )

# Also needed to allow rdf:li on all property element productions
# since we can't capture the rdf:li rewriting to rdf_<n> in relaxng

# Need to add these explicitly
xmllang = attribute xml:lang { text }
xmlbase = attribute xml:base { text }
# and to forbid every other xml:* attribute, element

doc =
  RDF | nodeElement

RDF =
  element rdf:RDF {
     xmllang?, xmlbase?, nodeElementList
}

nodeElementList =
  nodeElement*

  # Should be something like:
  #  ws* , (  nodeElement , ws* )*
  # but RELAXNG does this by default, ignoring whitespace separating tags.

nodeElement =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype | rdf:li |
```

```
                rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID ) {
      (idAttr | nodeIdAttr | aboutAttr )?, xmllang?, xmlbase?, propertyAttr*, propertyEltList
  }

  # It is not possible to say "and not things
  # beginning with _ in the rdf: namespace" in RELAX NG.

ws =
  " "

  # Not used in this RELAX NG schema; but should be any legal XML
  # whitespace defined by http://www.w3.org/TR/2000/REC-xml-20001006#NT-S


propertyEltList =
  propertyElt*

  # Should be something like:
  #  ws* , ( propertyElt , ws* )*
  # but RELAXNG does this by default, ignoring whitespace separating tags.

propertyElt =
  resourcePropertyElt |
  literalPropertyElt |
  parseTypeLiteralPropertyElt |
  parseTypeResourcePropertyElt |
  parseTypeCollectionPropertyElt |
  parseTypeOtherPropertyElt |
  emptyPropertyElt

resourcePropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr?, xmllang?, xmlbase?, nodeElement
  }

literalPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      (idAttr | datatypeAttr )?, xmllang?, xmlbase?, text
```

```
  }

parseTypeLiteralPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr?, parseLiteral, xmllang?, xmlbase?, literal
  }

parseTypeResourcePropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr?, parseResource, xmllang?, xmlbase?, propertyEltList
  }

parseTypeCollectionPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr?, xmllang?, xmlbase?, parseCollection, nodeElementList
  }

parseTypeOtherPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr?, xmllang?, xmlbase?, parseOther, any
  }

emptyPropertyElt =
   element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                 rdf:resource | rdf:nodeID | rdf:datatype |
                 rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                 xml:* ) {
       idAttr?, (resourceAttr | nodeIdAttr)?, xmllang?, xmlbase?, propertyAttr*
   }

idAttr =
  attribute rdf:ID {
```

```
      IDsymbol
  }

nodeIdAttr =
  attribute rdf:nodeID {
      IDsymbol
  }

aboutAttr =
  attribute rdf:about {
      URI-reference
  }

propertyAttr =
  attribute * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                  rdf:resource | rdf:nodeID | rdf:datatype | rdf:li |
                  rdf:Description | rdf:aboutEach |
                  rdf:aboutEachPrefix | rdf:bagID |
                  xml:* ) {
      string
  }

resourceAttr =
  attribute rdf:resource {
      URI-reference
  }

datatypeAttr =
  attribute rdf:datatype {
      URI-reference
  }

parseLiteral =
  attribute rdf:parseType {
      "Literal"
  }

parseResource =
  attribute rdf:parseType {
      "Resource"
  }

parseCollection =
  attribute rdf:parseType {
```

```
      "Collection"
  }

parseOther =
  attribute rdf:parseType {
      text
  }

URI-reference =
  string

literal =
  any

IDsymbol =
  xsd:NMTOKEN

any =
  mixed { element * { attribute * { text }*, any }* }
```

B Revisions since Draft 10 October 2003 (Informative)

Changes since 10 October 2003 second last call working draft

These are are divided into non-editorial and editorial. The non-editorial changes also list consquential editorial changes. Editorial changes are those which do not result in any change in the meaning of an RDF document or the behaviour of an RDF application.

Appendix B.1: Non-Editorial Revisions

None

Appendix B.2: Editorial Revisions

German Translation

Fix the German in Example 8 in section 2.7 after the comment by Benjamin Nowack

No property attributes on rdf:parseType="Resource"

2.5 Update to reflect the syntax definition that property attributes cannot be used with `rdf:parseType="Resource"` as pointed out by comment by Sabadello 2003-10-30

URI Encoding

6.1.6, 6.1.8, 6.1.9 after proposal by Jeremy Carroll
6.1.2, 6.1.4 Added element/attribute URI-string-value accessors
7.2.11, 7.2.21 Added use of new uri event for the `rdf:type` cases
7.2.11 (<e.URI> and <a.URI>), 7.2.15 (<e.URI>) 7.2.16 (<e.URI>) 7.2.17 (<e.URI>) 7.2.18 (<e.URI>) 7.2.19 (<e.URI> twice) 7.2.21 (<e.URI> twice, <a.URI> once) changed from X.URI to X.URI-string-value (anywhere "<"..">" appeared in the grammar action without a hardcoded URI reference)
7.2.32 Replace action wording with "An RDF URI reference"
All changed as outlined in proposal 2003-10-06 after comment by Patel-Schneider 2003-10-29

Appendix B.3: Issues requiring no document revisions

None